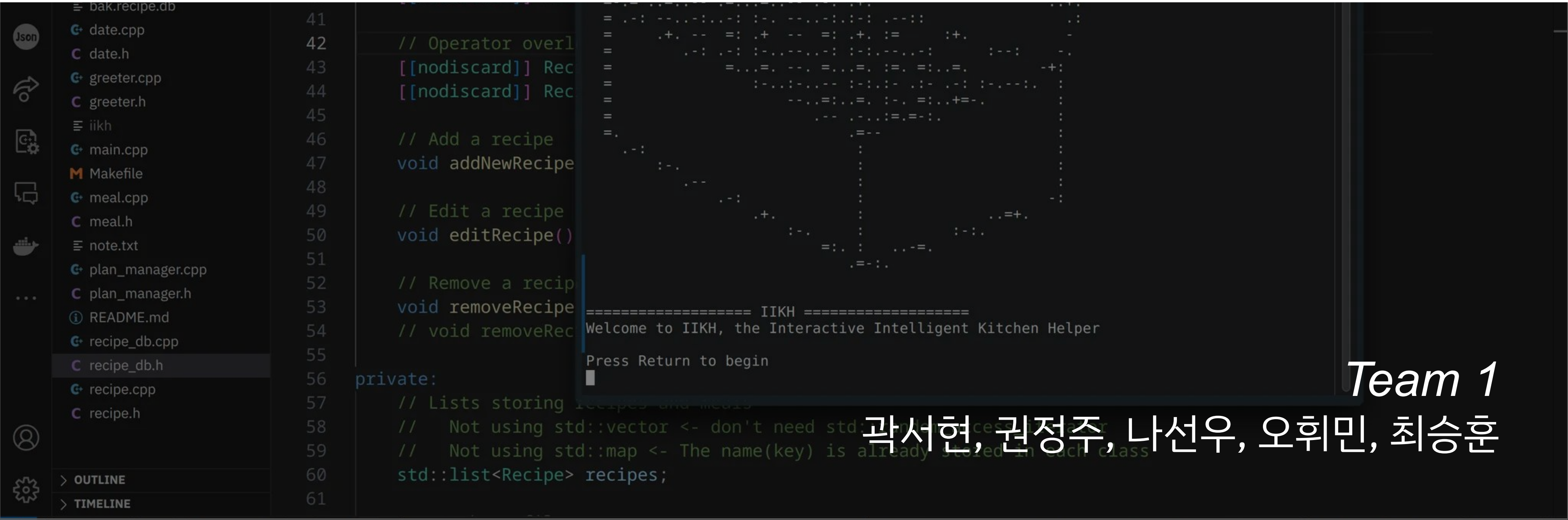
Object Oriented Programming [class 03]

IIKH Implementation



```
    // Operator overl
    [[nodiscard]] Rec
    [[nodiscard]] Rec

    // Add a recipe
    void addNewRecipe

    // Edit a recipe
    void editRecipe()

    // Remove a recip
    void removeRecipe
    // void removeRec

private:
    // Lists storing recipes and meals
    //   Not using std::vector <- don't need std:
    //   Not using std::map <- The name(key) is already stored in each class
    std::list<Recipe> recipes;
```

```
=================== IIKH ===================
Welcome to IIKH, the Interactive Intelligent Kitchen Helper

Press Return to begin
```

Team 1

곽서현, 권정주, 나선우, 오휘민, 최승훈

Class Designs

How classes in IIKH implemented

class Greeter

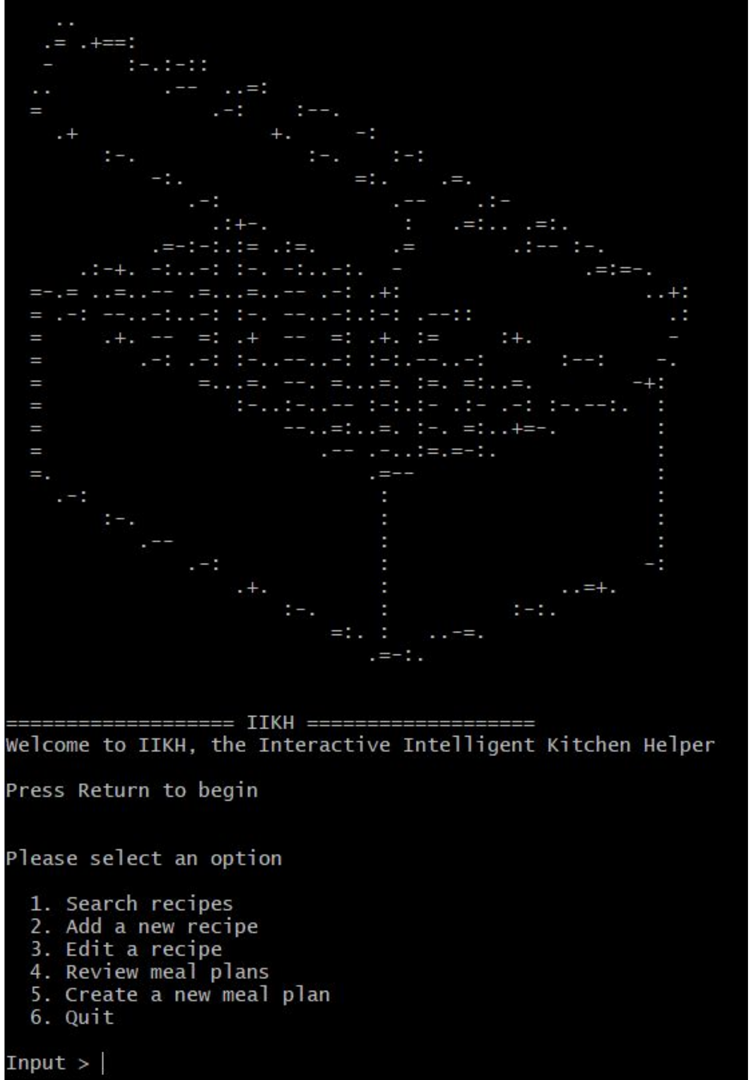
1 Print welcome messages

-'Welcome to the IIKH,
the Interactive Intelligent Kitchen Helper

-Press Return to begin'

2 Offer choice and send it to other classes

1. Search recipes
2. Add a new recipe
3. Edit a recipe
4. Review meal plans
5. Create a new plan of meals
6. Quit



1

2

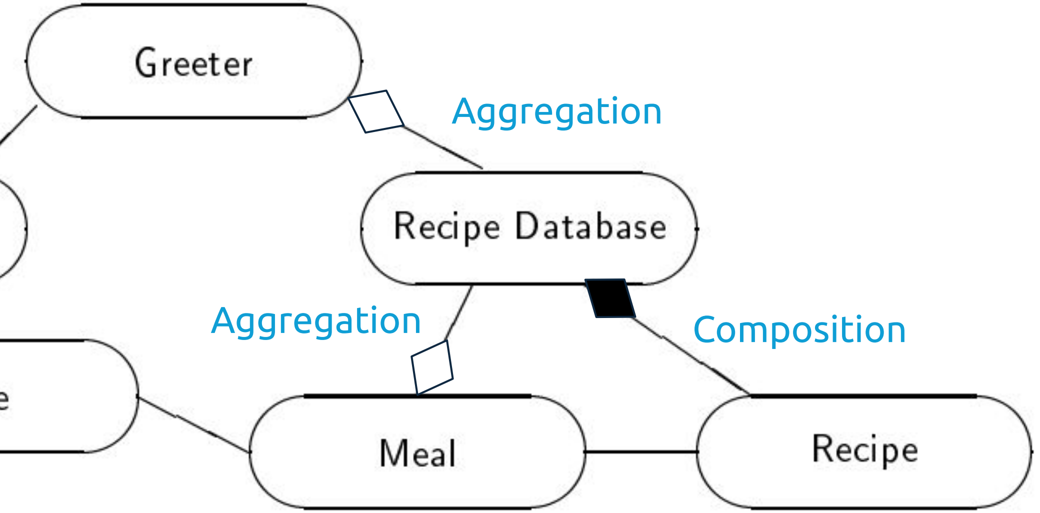
```
=================== IIKH ===================
Welcome to IIKH, the Interactive Intelligent Kitchen Helper

Press Return to begin


Please select an option

  1. Search recipes
  2. Add a new recipe
  3. Edit a recipe
  4. Review meal plans
  5. Create a new meal plan
  6. Quit

Input > |
```

Class RecipeDatabase

1. Constructor reads file using std::filesystem::path and std::filesystem.

2. searchRecipes(), addNewRecipe(), editRecipe()
   - interactively perform each operations.

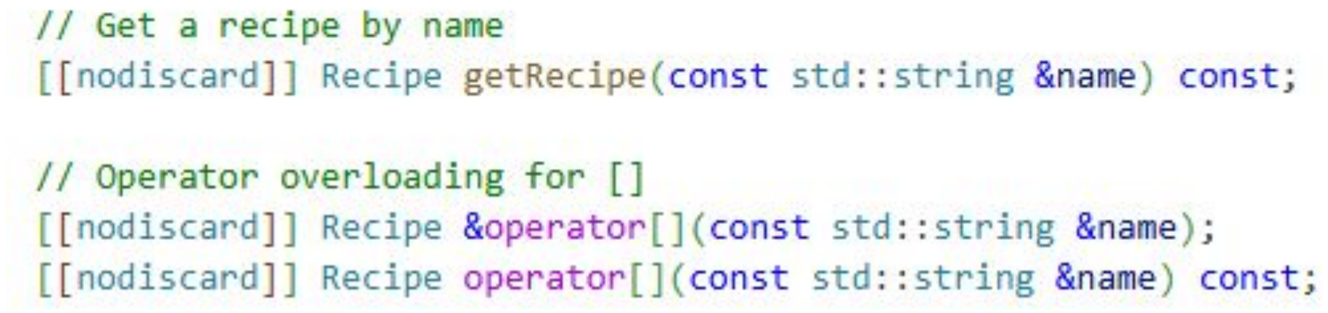
3. Destructor saves the database to the file.



- Stores Recipe list (std::list<T>)
- Stored in Greeter, Meal as a reference (RecipeDatabase &)

Class RecipeDatabase

Allows other classes to get a Recipe instance



```
// Get a recipe by name
[[nodiscard]] Recipe getRecipe(const std::string &name) const;

// Operator overloading for []
[[nodiscard]] Recipe &operator[](const std::string &name);
[[nodiscard]] Recipe operator[](const std::string &name) const;
```

Note: [[ ]] is an attribute specifier sequence (since C++11)

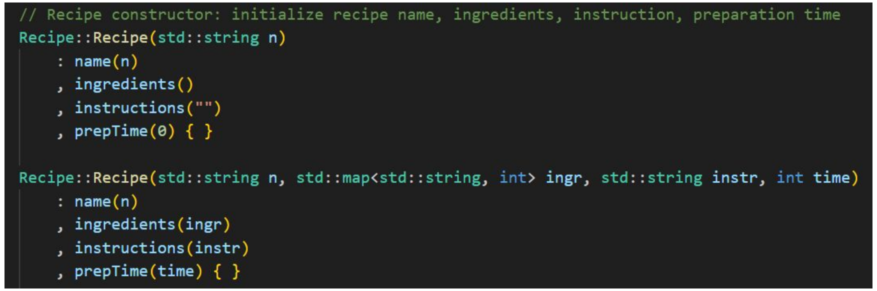
Overloaded function / Operations for convenience

i.e.) getRecipe() returns empty Recipe if name is not found,
while operator[] returns A REFERENCE to newly constructed object

class Recipe

1 Recipe constructor

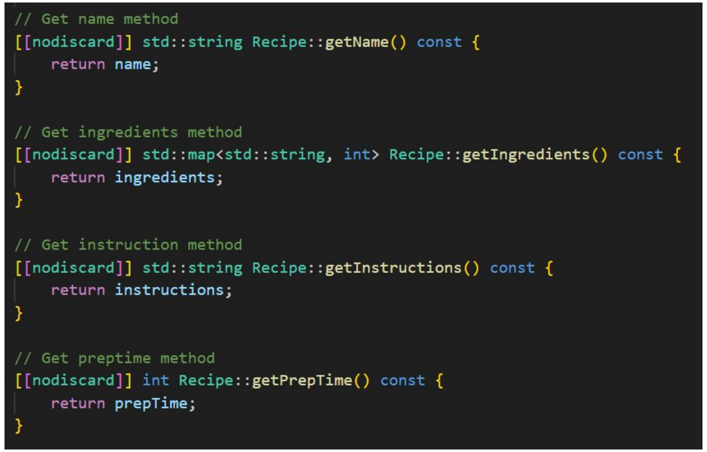
- initialize recipe name, ingredients, instruction, preparation time



```cpp
// Recipe constructor: initialize recipe name, ingredients, instruction, preparation time
Recipe::Recipe(std::string n)
    : name(n)
    , ingredients()
    , instructions("")
    , prepTime(0) { }

Recipe::Recipe(std::string n, std::map<std::string, int> ingr, std::string instr, int time)
    : name(n)
    , ingredients(ingr)
    , instructions(instr)
    , prepTime(time) { }
```

2 Getter method

- get name, instructions, ingredients, preptime

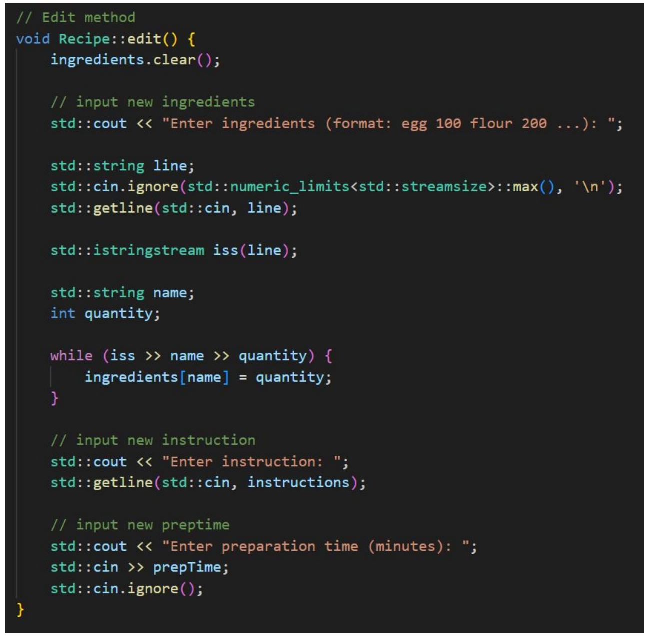


```cpp
// Get name method
[[nodiscard]] std::string Recipe::getName() const {
    return name;
}

// Get ingredients method
[[nodiscard]] std::map<std::string, int> Recipe::getIngredients() const {
    return ingredients;
}

// Get instruction method
[[nodiscard]] std::string Recipe::getInstructions() const {
    return instructions;
}

// Get preptime method
[[nodiscard]] int Recipe::getPrepTime() const {
    return prepTime;
}
```

class Recipe

3 Edit recipe

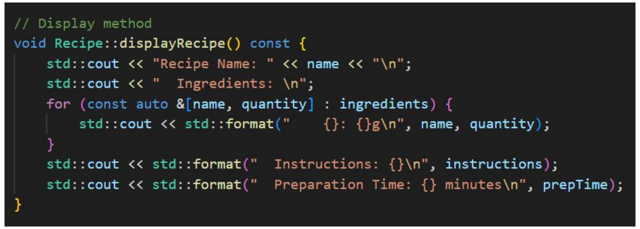
- edit ingredients, instructions, preptime



```cpp
// Edit method
void Recipe::edit() {
    ingredients.clear();

    // input new ingredients
    std::cout << "Enter ingredients (format: egg 100 flour 200 ...): ";

    std::string line;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::getline(std::cin, line);

    std::istringstream iss(line);

    std::string name;
    int quantity;

    while (iss >> name >> quantity) {
        ingredients[name] = quantity;
    }

    // input new instruction
    std::cout << "Enter instruction: ";
    std::getline(std::cin, instructions);

    // input new preptime
    std::cout << "Enter preparation time (minutes): ";
    std::cin >> prepTime;
    std::cin.ignore();
}
```

4 Display recipe

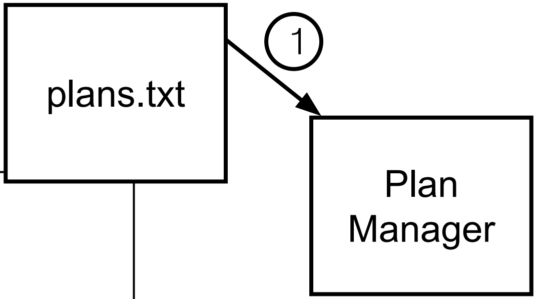
- display name, ingredients, instructions, preptime



```cpp
// Display method
void Recipe::displayRecipe() const {
    std::cout << "Recipe Name: " << name << "\n";
    std::cout << "  Ingredients: \n";
    for (const auto &[name, quantity] : ingredients) {
        std::cout << std::format("    {}: {}g\n", name, quantity);
    }
    std::cout << std::format("  Instructions: {}\n", instructions);
    std::cout << std::format("  Preparation Time: {} minutes\n", prepTime);
}
```

class PlanManager

1 Read all existing plans from "plans.txt"



-reads and loads all plans from a text file "plans.txt" and store in a list

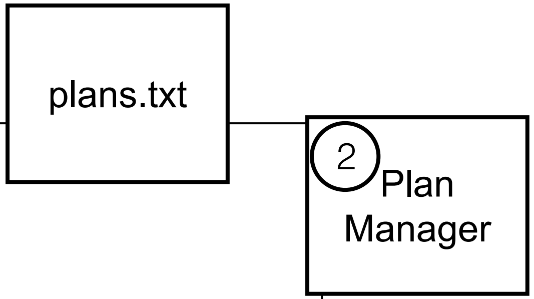
```
YYYY-MM-DD $#memo#$ [NameOfMeal]={NamesOfRecipes,serving}
```

-Memo is optional.

Using regex:
```
(\d{4}-\d{2}-\d{2})\$\#(.*?)\#\$\[(.*?)\]=\{([^}]+)\}
```

-Using Constructor: PlanManager()

2 reviewPlans(): show users all plans



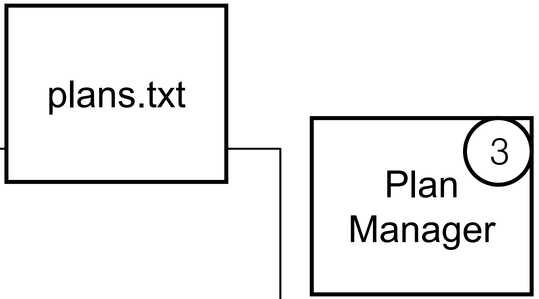
-Users can view the meal and recipe details for each plan.

-Able to add or remove recipes from the meal or change to memo.

-Call displayAndEdit(), displayMealInfo(), addRecipe(), or removeRecipe() to do certain tasks.

class PlanManager

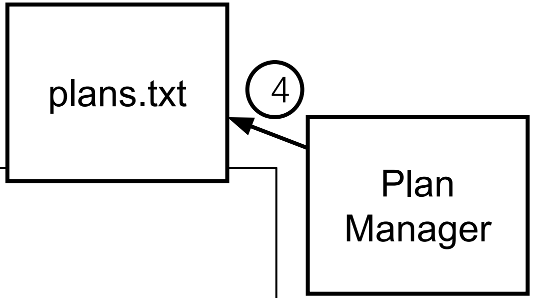
3 createNewPlan(): allow users to create a new plan and add some meals.



-Users can create a new meal plan by specifying the date and adding meals with their corresponding recipes.

-Call displayAndEdit(), manageMeals(), getMeals(), setMeals() to do certain tasks.

4 Write all plans into “plan.txt”, when the program terminates.



```
YYYY-MM-DD $#memo#$ [NameOfMeal]={NamesOfRecipes,serving}
```

-Using Destructor: ~PlanManager()

class Date

1 Constructor

Constructor Overloading

Minimizing code duplication and simplifying code through constructor chaining.

```cpp
Date(int year, int month, int day);
Date(const std::string &date);
Date(const std::string &date, const std::string &description);
Date(const std::string &date, const std::string &description, const std::list<Meal> &meals);
```

Return date (string), list of meals, and memo.

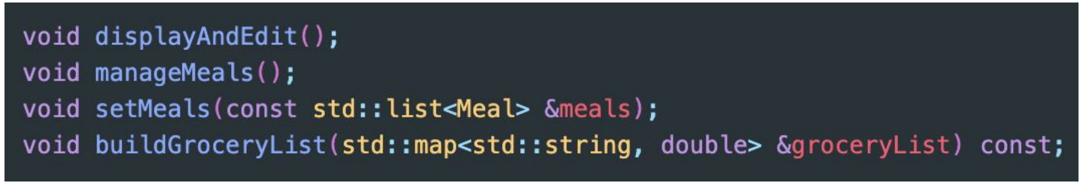
Getters 2

```cpp
[[nodiscard]] std::tuple<int, int, int> getDate() const;
[[nodiscard]] std::string getDateAsString() const;
[[nodiscard]] std::list<Meal> getMeals() const;
[[nodiscard]] std::string getMemo() const;
```

class Date

3 Important Methods

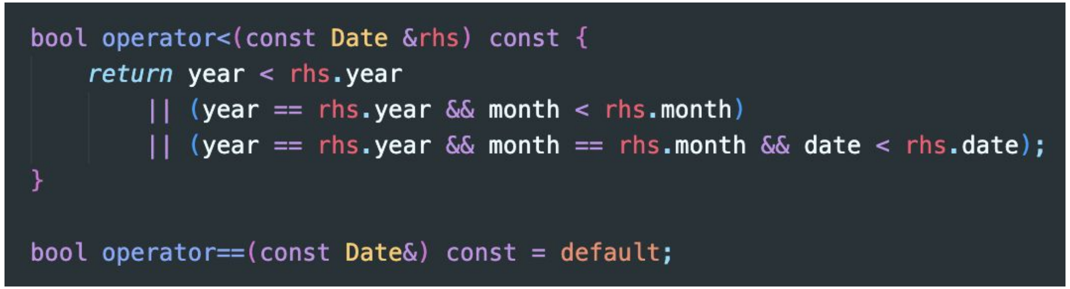
Functionality that manages meal plans and memos for specific dates, and generates a grocery list with the required ingredients and quantities based on the meal plans.



```
void displayAndEdit();
void manageMeals();
void setMeals(const std::list<Meal> &meals);
void buildGroceryList(std::map<std::string, double> &groceryList) const;
```

Operators 4

Date comparison operator overloading method.

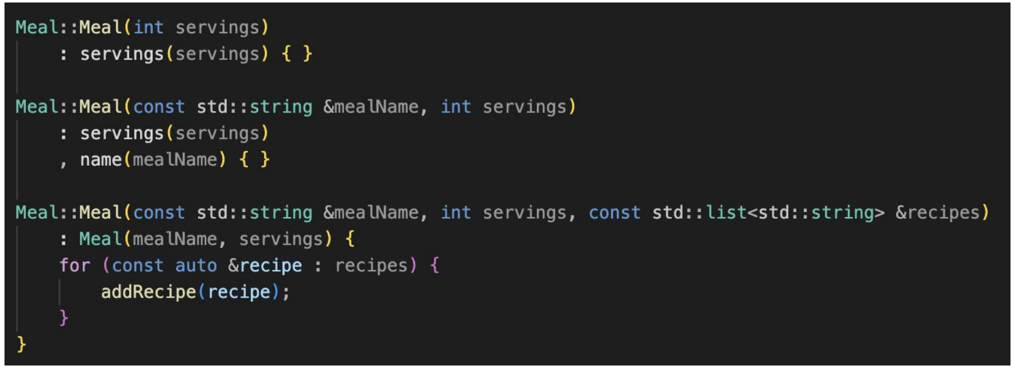


```
bool operator<(const Date &rhs) const {
    return year < rhs.year
        || (year == rhs.year && month < rhs.month)
        || (year == rhs.year && month == rhs.month && date < rhs.date);
}

bool operator==(const Date&) const = default;
```

class Meal

1 Constructor overloading

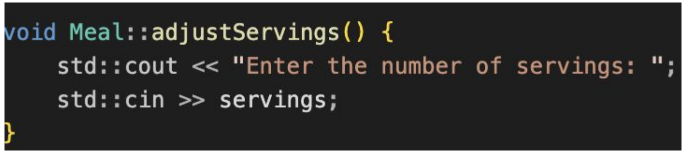
- allow creating Meal objects in multiple ways



```
Meal::Meal(int servings)
    : servings(servings) { }

Meal::Meal(const std::string &mealName, int servings)
    : servings(servings)
    , name(mealName) { }

Meal::Meal(const std::string &mealName, int servings, const std::list<std::string> &recipes)
    : Meal(mealName, servings) {
    for (const auto &recipe : recipes) {
        addRecipe(recipe);
    }
}
```

2 adjustServings method

- prompt users to input the number of servings

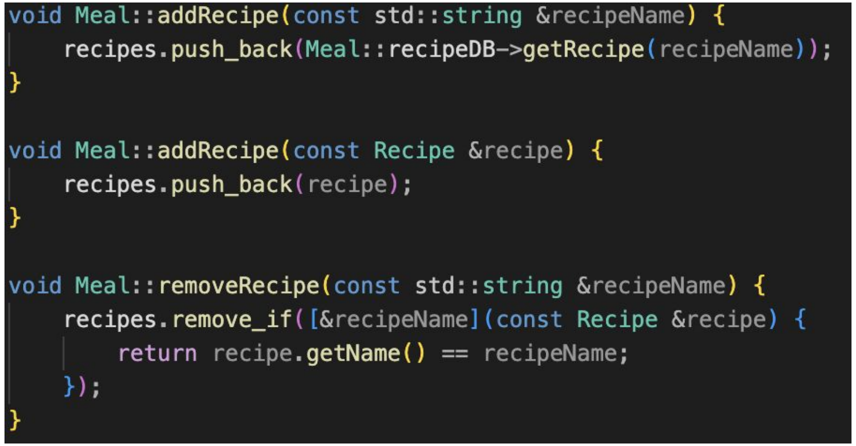


```
void Meal::adjustServings() {
    std::cout << "Enter the number of servings: ";
    std::cin >> servings;
}
```

class Meal

3 add & remove recipe

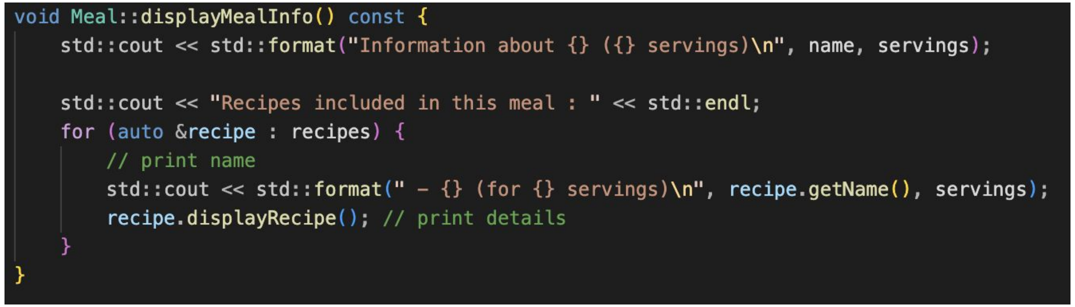
- add recipe to list ‘recipes’
- remove recipe from list ‘recipes’



```cpp
void Meal::addRecipe(const std::string &recipeName) {
    recipes.push_back(Meal::recipeDB->getRecipe(recipeName));
}

void Meal::addRecipe(const Recipe &recipe) {
    recipes.push_back(recipe);
}

void Meal::removeRecipe(const std::string &recipeName) {
    recipes.remove_if([&recipeName](const Recipe &recipe) {
        return recipe.getName() == recipeName;
    });
}
```

4 displayMealInfo method

- display information about meal



```cpp
void Meal::displayMealInfo() const {
    std::cout << std::format("Information about {} ({} servings)\n", name, servings);

    std::cout << "Recipes included in this meal : " << std::endl;
    for (auto &recipe : recipes) {
        // print name
        std::cout << std::format(" - {} (for {} servings)\n", recipe.getName(), servings);
        recipe.displayRecipe(); // print details
    }
}
```

Special Points

With powerful, modern C++ grammars & features

Special Points

[[nodiscard]]

emplace_back

std::format("{}", "hi")

structured bindings

```
auto &[k, v] = map
```

static_assert

R-Value reference (&&)

constexpr

std::filesystem

Demonstration

Please refer to external video